Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних та інформаційних технологій
(повне найменування інституту, назва факультета (відділення))

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної, цикловоі комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: «Розроблення аплікації автоматичного перетворення звукового запису у текстовий формат з використанням Google Cloud Speech-to-Text та Spring Boot»

Виконав студент 2 курсу, групи ІСТС-21
спеціальності:
126 „Інформаційні системи та технології"
(шифр і назва напрямку підготовки спеціальності)

Бокало Н.Д.
(прізвище та ініціали)

Керівник *Івасюк Р.В.*
Павлюк У.В.
(прізвище та ініціали)

Рецензент Крошний І.М.
(прізвище та ініціали)

Львів - 2024

Національний лісотехнічний університет України

(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій

Кафедра інформаційних систем та комп'ютерного моделювання

Рівень вищої освіти перший (бакалаврський)

Спеціальність 126 „Інформаційні системи та технології"

ЗАТВЕРДЖУЮ

Завідувач кафедри ІСКМ

_____ Сторожук О.Л.

"06" 02 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Бокалу Назару Дмитровичу

(прізвище, ім'я, по батькові)

1.Тема роботи: Розроблення аплікації автоматичного перетворення звукового запису у текстовий формат з використанням Google Cloud Speech-to-Text та Spring Boot.

Керівник проекту асистент Івасюк Руслан Васильович

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від 06.02.2024 р. № С-87

2. Термін подання студентом роботи 10.06.2024. р.

3. Вихідні дані до роботи *Розробити вебдодаток для перетворення аудіозапису на текст за допомогою фреймворку Spring Boot та Google Api speech to text. Розробити BackEnd частину для обробки даних. Реалізувати зручний та простий інтерфейс для адмін і користувацької частини.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) Стан проблемної області
2) Інформаційне забезпечення
3) Програмне забезпечення
4) Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Підготовка презентаційного матеріалу до доповіді (слайди для доповіді).

6. Дата видачі завдання 7 лютого 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№, з/п	Етапи бакалаврської роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	08.02.2023-12.02.2024	Виконано
2.	Постановка задачі і її формалізація	13.02.2024-17.02.2024	Виконано
3.	Виконання вхідного етапу технології	20.02.2024-27.02.2024	Виконано
4.	Реалізація головних алгоритмів проекту	03.03.2024-20.03.2024	Виконано
5.	Виконання етапу відлагодження проекту	22.03.2024-09.04.2024	Виконано
6.	Тестування проекту	10.04.2024-12.04.2024	Виконано
7.	Оформлення записки до дипломного проекту.	05.05.2024-14.06.2024	Виконано

Студент _____ Бокало Н.Д.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Івасюк Р.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Павлюк У.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 48 сторінок пояснювальної записки, 33 рисунків, 2 додатки, 15 джерел.

Дипломна робота присвячена розробці вебдодатку для перетворення мовлення в текст. Додатковим сервісом виступає можливість авторизованим користувачам переглядати та редагувати записи. Робота написана об’єктно орієнтованою мовою Java. Під час розробки вебзастосунку було використано декілька фреймворків, а саме Spring Boot як основа розробки веб застосунку, Spring Security для авторизації та аутентифікації, Google API Speech-to-Text для перетворення аудіо в текст, також Spring Data Jpa для комфортного з'єднання з базою даних, React для побудови користувацького інтерфейсу.

Ключові слова: Java, Spring Boot, Spring Security, JavaScript, Google API Speech-to-Text, React, MongoDB, Spring Data Jpa.

ABSTRACT

The thesis contains 48 pages of explanatory note, 33 figures, 2 appendices, 15 sources.

The thesis is devoted to the development of a web application for converting speech into text. An additional service is the ability for authorized users to view and edit records. The work is written in the object-oriented Java language. During the development of the web application, several frameworks were used, namely Spring Boot as the basis for the development of the web application, Spring Security for authorization and authentication, Google API Speech-to-Text for converting audio to text, as well as Spring Data Jpa for a comfortable connection with a database, React to build the user interface.

Keywords: Java, Spring Boot, Spring Security, JavaScript, Google API Speech-to-Text, React, MongoDB, Spring Data Jpa.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити вебдодаток для перетворення аудіо запису на текст. Створити базу даних з інформацією про користувачів, записок.

Використати стек технологій: мови програмування Java, MongoDB для розробки бази даних та технологію Spring Data Jpa для комфортної взаємодії з базою даних, Google cloud speech to text для перетворення mp3 в текстовий формат, фреймворк Spring Security для реєстрації та надання ролі користувачу, Spring Boot для об’єднання та конфігурування всіх модулів, JavaScript, React для спрощення процесу створення складних інтерфейсів.

Розробити серверну і клієнтську частини, що мають забезпечувати виконання наступних вимог:

1. Реалізація простого, зручного і зрозумілого інтерфейсу;
2. Можливість записувати аудіо голос та переглядати у текстовому форматі;
3. Можливість переглядати, видаляти, редагувати збережені записи;
4. Можливість здійснення запису тільки для зареєстрованих та залогінених користувачів;
5. Реалізація логінації для входу в адмін режим;
6. У режимі адміністратора передбачити можливість для таких операцій:
    - Переглядати записи всіх користувачів;
    - Додавання, видалення та редагування інформації про записи;
    - Переглядати всіх зареєстрованих користувачів;

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

JPA - стандарт Java для роботи з об'єктно-реляційним відображенням даних в програмах.

SDK - набір засобів для розробки програмного забезпечення.

MVC - model-view-controller - паттерн проектування.

DTO - шаблон проектування в програмуванні, для передачі даних.

JWT "Json Web Token" - використовується для автентифікації та передачі даних.

ВСТУП

У сучасному світі інформаційні технології відіграють ключову роль у полегшенні і покращенні способів взаємодії людини з комп'ютерами та електронними пристроями. Одним із напрямків активного розвитку є системи автоматичного розпізнавання мовлення та його перетворення у текстовий вигляд. Ця технологія має великий потенціал у різних сферах, від комп'ютерного спілкування до медичної документації.

Об’єктом дослідження виступає інформаційна система, яку необхідно розробити.

Предметом дослідження є реалізація взаємодії між фронтенд та бекенд частинами проекту та застосування функціональних можливостей сторонніх бібліотек вебдодатку.

Метою роботи є розробка інформаційного вебдодатка засобами фреймворку Spring Boot, який би надавав можливості для запису аудіо запису та перегляду текстовій формі.

Завдання розробити веб-додатка для перетворення аудіо запису на текст. Створити базу даних з інформацією про користувачів, записок.

Практичне значення розроблений програмний продукт можна икористовувати для швидкого та зручного створення записок за допомоги мовлення та переглядати збережені записи у зручному форматі.

РОЗДІЛ 1. ОГЛЯД ПРОБЛЕМНОЇ ОБЛАСТІ

Чи може розробка перетворення мовлення в текст бути корисною? Так, розробка перетворення мовлення в текст може допомогти користувачам, підприємствам покращити їхню ефективність у багатьох аспектах. Ось декілька прикладів:

- Підвищення продуктивності: Перетворення мовлення в текст може автоматизувати багато завдань, які потребують ручного введення тексту, таких як диктування електронних листів, запис нотаток та транскрипція аудіозаписів. Це може звільнити час працівників для більш важливих завдань.
- Підвищення доступності: Перетворення мовлення в текст може зробити контент доступним для людей з обмеженими можливостями, таких як люди з порушеннями слуху або люди, які не можуть читати. Це може розширити коло потенційних клієнтів та співробітників.
- Розширення аудиторії: Це може розширити коло потенційних клієнтів, студентів, співробітників та інших зацікавлених сторін, що може принести користь як користувачам, так і підприємствам.
- Підвищення інновацій: Перетворення мовлення в текст може використовуватися для створення нових продуктів і послуг, які раніше були неможливими.

Розпізнавання мови – це вебдодаток, призначений для перетворення аудіо сигналу на текст. Цей веб-додаток мови має широкий спектр застосувань, таких як:

- Диктування тексту
- Перегляд збережених записів
- Видалення старих записів
- Автоматизація обслуговування користувача
- Котроль зі сторони адміну
- Редагування збережених записів

Розробка перетворення мовлення в текст має значний потенціал для покращення ефективності роботи. Ця технологія може автоматизувати багато завдань. Підприємства, які розглядають можливість використання цієї технології, повинні ретельно дослідити її переваги та недоліки. Загалом, розробка перетворення мовлення в текст може бути цінним інструментом для підприємств, які прагнуть покращити свою ефективність. Важливо використовувати цю технологію з розумом і відповідально, щоб вона приносила максимальну користь вашій справі.

Додаток побудований на Spring Boot, який розбивається на логічні модулі. Контролер має бути дуже простим. Щоб контролер діяв як координатор і делегат, а не виконував фактичну бізнес-логіку.

За замовчуванням контролери є одиночними, і будь-який стан може спричинити багато проблем. Контролер не повинен виконувати бізнес-логіку, а покладатися на делегування. Контролер повинен обробляти HTTP-рівень програми, це не слід передавати службі. Контролери повинні бути розроблені відповідно до варіантів використання можливостей. Сервіс — ще одна основна концепція Spring Boot. Структурувати послуги навколо бізнес-функцій. Розробка служб із такими іменами, як NoteService, UserService. Використовувати однозначне відображення між контролером і службою, це є хороша практика.

База даних незалежна від основної бізнес-логіки. Рівень бази даних розташований, щоб логіка бази даних була відокремлена від служби, ми не хочемо, щоб служба знала, з якою базою даних вона спілкується, що вимагає певної абстракції для інкапсуляції збереження об’єкта.

Введення конструктора - це є хорошою практикою. Один із способів захистити вашу бізнес-логіку від коду Spring Boot — це використання ін’єкції конструктора. @Autowired annotation не тільки необов’язкова для конструктора, але вона також полегшує створення екземплярів bean-компонентів.

Забезпечення глобальної обробки винятків. Потрібен послідовний спосіб обробки винятків. Spring Boot пропонує два основні підходи: Ви повинні використовувати HandlerExceptionResolver, щоб визначити глобальну стратегію обробки винятків; Ви також можете додати анотацію @ExceptionHandler до контролера, яка може бути корисною в деяких конкретних сценаріях.

Перевірка коду, якщо ви не пишете тести, ви пишете застарілий код із самого початку. Якщо хтось інший використовує ваш код, змінювати будь-що там стане небезпечно. Це може бути ще більш ризикованим, якщо у вас є кілька служб, які залежать одна від одної. Використовувати тестові зрізи, щоб зробити тестування легшим і більш зосередженим. Тестування коду за допомогою Spring Boot може бути складним — вам потрібно ініціалізувати рівень даних, підключити багато служб. Відповідь полягає в тому, щоб використовувати тестові зрізи. За допомогою тестових фрагментів ви можете підключати лише частини додатка за потреби. Це може заощадити вам багато часу та гарантувати, що ваші тести не будуть пов’язані з невикористаним вмістом.

У веб-розробці REST API відіграють важливу роль у забезпеченні безперебійного зв’язку між клієнтом і сервером. Зв’язок між клієнтом і сервером зазвичай не надто прямий. Тому ми використовуємо інтерфейс, який називається інтерфейсом прикладного програмування, щоб діяти як посередник між клієнтом і сервером. Оскільки API відіграє вирішальну роль у цьому зв’язку між клієнтом і сервером, ми завжди повинні розробляти API з урахуванням найкращих практик. REST це архітектурний стиль програмного забезпечення, створений Роєм Філдінгом у 2000 році для керівництва проектуванням архітектури для Інтернету. Будь-який API (інтерфейс прикладного програмування), який відповідає принципу дизайну REST, називається RESTful. Простіше кажучи, REST API є середовищем для двох комп’ютерів для обміну даними через HTTP (протокол передачі гіпертексту), таким же чином, як спілкуються клієнти та сервери. У минулому приймалися та відповідали на них переважно в XML і HTML. Але сьогодні JSON став

форматом для надсилання та отримання даних. Це пов’язано з тим, що, наприклад, у XML часто декодувати та кодувати дані – тому XML більше не підтримується фреймворками. JavaScript, наприклад, має вбудований метод для аналізу даних JSON через API отримання, оскільки JSON в першу чергу створювався для нього. Але якщо ви використовуєте будь-яку іншу мову програмування, усі вони тепер також мають методи аналізу та обробки даних JSON.

Загалом завдяки Spring Boot створювати додатки простіше. Завдяки цим найкращим практикам ваш процес впровадження не тільки стане швидшим, але й стане надійнішим і успішнішим у довгостроковій перспективі.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Java

Існує багато мов програмування та фактична кількість невідома. Однак приблизно одна десята частина мов програмування використовується в сучасній IT-індустрії, наприклад, при розробці програмного забезпечення та розробці програм. Одна з таких мов є Java. Java більш ніж 10 років є одна із самих затребуваних мов програмування на IT-ринку.

Java - це мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems».

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Головним мотивом створення Java була потреба в мові програмування, яка б не залежала від платформи (тобто від архітектури) і яку можна було б використовувати для створення програмного забезпечення, що вбудовується в різноманітні побутові електронні прилади, такі як мобільні засоби зв'язку, пристрої дистанційного керування тощо.

Чому саме Java? Java має багато плюсів про які зараз поговоримо.

Типи даних - java є суворо типізованою мовою, кожна змінна та вираз має тип, відомий на етапі компіляції. Типи даних Java належать до двох категорій: прості (primitive) та вказівникові (reference). До простих типів належить булевий(логічний) тип, числові типи та символьний тип. Числові типи складаються із цілих типів byte, short, int, long та дійсних типів float, double. Символьний тип представлений типом char. Вказівникові типи складаються із класів, інтерфейсів, масивів. Значенням вказівникового типу є вказівник на об'єкт — екземпляр класу чи масиву. Рядки є об'єктами класу String. Отже за рахунок строгої типізації програмісту стає легше читати код і передбачати можливі помилки у коді [4, с.18].

ООП - Java слідує парадигмі ООП (об'єктно-орієнтованого програмування), що полегшує створення модульних програм і коду для багаторазового

використання. Це сприяє розробці простіших, легших в обслуговуванні кодових баз для масштабних проектів.

Кросплатформеність - кросплатформеність в Java означає здатність програм, написаних на Java, працювати на різних операційних системах без змін вихідного коду. Основна ідея кросплатформеності в Java полягає в тому, що ви можете написати програму один раз, скомпілювати її в байт-код Java, а потім виконувати цей байт-код на будь-якому пристрої або операційній системі. Ви не потребуєте різних версій програми для Windows, macOS, Linux тощо.

Застосування в різних галузях - java широко використовується в різних сферах програмування, таких як веб-розробка, мобільна розробка (Android), розробка десктопних додатків, розробка серверних додатків, інтеграція підприємств, ігрова розробка та багато іншого.

Спільнота - java має велику спільноту розробників, що означає наявність великої кількості ресурсів, бібліотек та фреймворків, які допомагають у процесі розробки. Ця мережа підтримки може бути безцінною для програмістів, які шукають допомогу, поради чи співпрацю.

Розробка веб-додатків на Java є популярним напрямом і використовується для створення різноманітних типів веб-застосунків, включаючи корпоративні системи, електронні комерційні платформи, соціальні мережі, інтернет-магазини та багато іншого. Основні компоненти розробки веб-додатків на Java включають: фреймворки(Spring), протоколи, інструменти для розробки, різні бібліотеки[15].

Розробка мобільних додатків для платформи Android також може проводитися з використанням Java. Android підтримує розробку додатків з використанням мови Java, яка добре інтегрується з Android SDK (Software Development Kit) і забезпечує широкі можливості для створення різноманітних мобільних додатків. Основні аспекти розробки мобільних додатків для Android з використанням Java включають: Android SDK, який містить набір інструментів, бібліотек і фреймворків для побудови Android-додатків. Приклади

популярних мобільних додатків для платформи Android, які були розроблені з використанням Java: Instagram, Facebook, YouTube.

Розробка десктопних додатків на Java є популярним напрямом і дозволяє створювати потужні, багатофункціональні програми для роботи на комп'ютерах з різними операційними системами, такими як Windows, macOS або Linux. Основні аспекти розробки десктопних додатків на Java включають: Java Swing або JavaFX: Для створення графічного інтерфейсу користувача (GUI) в десктопних додатках на Java зазвичай використовуються бібліотеки Swing або більш сучасний JavaFX. Ці бібліотеки дозволяють легко створювати вікна, кнопки, тексти, таблиці та інші елементи інтерфейсу. Приклади популярних десктопних програм, які були розроблені з використанням Java: Eclipse, IntellIj.

2.2 Фреймворк Spring Boot

Spring Boot - це фреймворк для розробки Java додатків, який забезпечує швидке створення самостійних, готових до використання додатків на основі платформи Spring [6]. Основна ідея Spring Boot полягає в спрощенні конфігурації та розгортання додатків, забезпечуючи заздалегідь налаштовані стартові умови, що дозволяють розробникам швидко створювати додатки без необхідності вручну налаштовувати велику кількість компонентів.

Основні особливості Spring Boot:

- Конвенція над конфігурацією: Spring Boot надає заздалегідь налаштовані конфігураційні параметри, що дозволяють створювати додатки з меншим обсягом коду конфігурації.
- Вбудований контейнер додатків: Spring Boot містить вбудовані сервери додатків (наприклад, Tomcat, Jetty або Undertow), що дозволяє створювати самостійні додатки, які можна запустити просто запустивши JAR-файл.
- Управління залежностями: Spring Boot використовує систему автоматичного управління залежностями Maven або Gradle для автоматичного вирішення залежностей та включення необхідних бібліотек.

- Автоматична конфігурація: Spring Boot надає автоматичну конфігурацію для багатьох бібліотек та компонентів Spring, що дозволяє автоматично налаштовувати додатки на основі класифікації класів та анотацій.
- Підтримка вбудованих баз даних: Spring Boot дозволяє створювати додатки, які використовують вбудовані бази даних (наприклад H2), спрощуючи процес розгортання та тестування.

Переваги впровадження Spring Boot:

У цілому, Spring Boot пропонує зручний і стандартизований спосіб створення Java додатків, зменшуючи кількість необхідного коду та спрощуючи процес розробки, конфігурації та розгортання додатків. Spring Boot виділяється серед інших фреймворків, оскільки він надає розробникам програмного забезпечення гнучке налаштування, надійну пакетну обробку, ефективний робочий процес та велику кількість інструментів, допомагаючи розробляти надійні та масштабовані програми на базі Spring.

Також слід зазначити що Spring Boot включає в себе MVC(Model View Controller) - це архітектурний шаблон, що використовується в програмному забезпеченні для розділення компонентів додатка на три основні частини: Модель, Вид і Контролер. Цей шаблон допомагає створити добре організовану структуру програми, де кожна частина відповідає за свої власні обов'язки [1 с. 549]. Загалом, шаблон MVC є добре встановленим підходом до розробки програмного забезпечення, який допомагає підтримувати структурований і ефективний код, зменшуючи залежність між різними компонентами додатка.

2.3 Spring Security

Перед тим як розпочати працювати з Spring Security, необхідно знати, що таке автентифікацію та авторизація.

Автентифікація визначає процес перевірки того, що суб'єкт (наприклад, користувач) дійсно є тим, ким він стверджує, що є. Це означає, що користувач повинен підтвердити свою ідентичність, наприклад, за допомогою логіна і пароля, біометричних даних, ключа безпеки або іншого фактора ідентифікації.

Після успішної автентифікації система може визнати користувача як дійсного і дозволити доступ до ресурсів чи функціоналу [13].

Авторизація визначає процес визначення того, чи має дозвіл користувач на доступ до певних ресурсів або виконання певних дій. Після того, як користувач успішно автентифікований, система визначає, які дії він може виконати і які ресурси він може переглядати чи змінювати. Це виконується на основі встановлених правил і політик безпеки, таких як ролі, дозволи, обмеження доступу тощо.

Spring Security - це фреймворк для забезпечення безпеки додатків на основі Spring. Вона надає широкі можливості для автентифікації, авторизації і захисту веб-додатків і мікросервісів. Основна мета Spring Security полягає в тому, щоб забезпечити надійний механізм захисту для застосунків Java [5].

Основні функціональні можливості Spring Security включають:

- Автентифікація: Spring Security дозволяє налаштовувати різні механізми автентифікації, такі як базова аутентифікація, форма входу, аутентифікація на основі токенів і т. д. Це дозволяє підтверджувати ідентичність користувачів.
- Авторизація: Spring Security дозволяє визначати права доступу для різних частин додатку на основі інформації про автентифікованих користувачів. Можливості включають визначення ролей, виразів доступу і анотацій для обмеження доступу до методів.
- Захист веб-додатків: Spring Security надає фільтри безпеки, які можна легко інтегрувати в веб-додатки Spring.
- Інтеграція з існуючими технологіями: Spring Security підтримує інтеграцію з іншими рішеннями Spring, такими як Spring Boot, Spring MVC, Spring Data і т. д. Вона також може легко інтегруватися з різними системами іншого роду за допомогою розширень і налаштувань.

В цілому, Spring Security дозволяє розробникам створювати безпечні, захищені додатки з різними рівнями контролю доступу і механізмів автентифікації.

2.4 JavaScript

JavaScript (JS) — динамічна, об'єктно-орієнтована прототипна мова програмування. Найчастіше використовується для створення веб сторінок, що надає можливість на боці клієнта взаємодіяти з користувачем, керувати браузером, обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб сторінки.

JavaScript, наразі, є однією з найпопулярніших мов програмування в інтернеті. В перші роки існування, більшість професійних програмістів скептично ставилися до мови, але її подальші модифікації внесли багато корисних можливостей, яких не вистачало для ефективного програмування. В результаті, були розроблені та покращені багато практик використання JavaScript (зокрема, тестування та налагодження), створені бібліотеки та фреймворки, поширилося використання JavaScript поза браузером [14]. В 2023 році, у категорії найпопулярніших мов програмування серед IT-спеціалістів згідно рейтингу IEEE Spectrum, який охоплював 59 мов програмування, топ-5 виглядав так: Python, Java, C++, C, JavaScript. JavaScript має низку властивостей об'єктно-орієнтованої мови. Крім того, JavaScript має кілька властивостей, притаманних функціональним мовам, — функції як об'єкти першого класу.

JavaScript має C-подібний синтаксис, але в порівнянні з мовою C має такі корінні відмінності: автоматичне приведення типів, анонімні та стрілочні функції, автоматичне збирання сміття, обробка винятків.

Ось кілька цікавих фактів про JavaScript:

- Загальність: JavaScript є однією з найпопулярніших мов програмування у світі. Вона підтримується майже всіма сучасними браузерами і є стандартом веб-розробки.
- Динамічна типізація: JavaScript є мовою зі слабкою динамічною типізацією. Це означає, що ви можете використовувати змінні без вказання типу даних, і типи даних можуть автоматично змінюватися під час виконання програми.

- Функціональна і об'єктно-орієнтована парадигми: JavaScript підтримує як функціональне програмування (за допомогою замикань, функцій вищих порядків тощо), так і об'єктно-орієнтоване програмування (за допомогою об'єктів, прототипів).
- Асинхронність: JavaScript використовує події та зворотні виклики (callbacks) для обробки асинхронних подій. Це робить його особливо корисним для розробки клієнтських веб-додатків, які повинні реагувати на дії користувача без блокування інтерфейсу.
- Використання на сервері: Поза веб-розробкою, JavaScript також широко використовується для створення серверних додатків за допомогою платформи Node.js. Node.js дозволяє розробникам створювати швидкі і ефективні сервери, використовуючи JavaScript.

Загалом JavaScript є потужною мовою програмування, яка широко використовується у світі програмування. Вивчення JavaScript відкриє перед вами безліч можливостей для створення різноманітних додатків і проектів.

2.5 Бібліотека React

React (також React.js, ReactJS) — відкрита JavaScript бібліотека для створення інтерфейсів користувача. Розробляється Meta (раніше Facebook) і спільнотою індивідуальних розробників. Бібліотеку створено Джорданом Волком, програмістом з Facebook. React дозволяє розробникам створювати великі веб застосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, маштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб застосунків. Як бібліотеку інтерфейсу

користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux [10].

Основні особливості React:

- Компонентна модель: React базується на концепції компонентів. Кожен елемент інтерфейсу є окремим компонентом, який має свій стан (state) і може відображати дані. Компоненти можна компонувати один в одного для створення складних інтерфейсів.
- Virtual DOM: React використовує віртуальний DOM (Document Object Model), щоб забезпечити ефективне оновлення інтерфейсу. Замість безпосередньої модифікації реального DOM при зміні даних, React спочатку вносить зміни у віртуальний DOM і порівнює його з реальним DOM. Після цього React виконує мінімальне число операцій для оновлення реального DOM, що дозволяє підвищити продуктивність додатка.
- JSX (JavaScript XML): React використовує JSX для опису структури інтерфейсу у вигляді розмітки, що схожа на HTML. JSX дозволяє вбудовувати JavaScript у розмітку, що полегшує створення динамічного контенту та компонування компонентів.
- Односторінкові додатки (Single Page Applications, SPA): React добре підходить для створення односторінкових додатків, де весь контент завантажується раз і далі оновлюється динамічно без повторного завантаження сторінки.

Ця бібліотека стала дуже популярною серед розробників завдяки своїй продуктивності, зручності у використанні та активному співтовариству.

2.6 Поняття Spring Data Jpa

Spring Data JPA — це частина Spring Framework, яка надає розширення для роботи з базами даних. Вона дозволяє зменшити багато повторюваного коду, який зазвичай пов'язаний з розробкою доступу до даних, шляхом автоматизації багатьох стандартних операцій [7].

Варто зазначити, що Spring Data Jpa є специфікація, а Hibernate - це його популярна реалізація. Специфікація - це набір певних класів та інтерфейсів для роботи та з’єднання з базою даних [12].

Hibernate - це ORM фреймворк. ORM(object relational mapping) - процес перетворення об’єктно-орієнтованої моделі в реляційну і навпаки [12].

Основні поняття та функції Spring Data JPA включають:

- Репозиторії: Spring Data JPA надає анотації та інтерфейси для визначення репозиторіїв, які автоматично генеруються під час виконання. Це означає, що вам не потрібно писати рутинний код для створення об'єктів доступу до даних.
- Методи запитів: Spring Data JPA дозволяє описувати запити до бази даних за допомогою іменованих методів в інтерфейсах репозиторіїв. Наприклад, ви можете оголосити метод findByLastName(String lastName), і Spring Data JPA автоматично створить запит для пошуку записів за прізвищем.
- Автоматична реалізація методів: Багато стандартних операцій, таких як збереження, оновлення, видалення, а також пошук за ідентифікатором, автоматично реалізуються Spring Data JPA, що полегшує розробку.

Spring Data JPA спрощує розробку шару доступу до даних в Spring-програмах. Spring Data JPA позбавляє програміста необхідності писати рутинний код і дозволяє витрачати більше часу на розробку бізнес-логіки та функціональності додатку.

2.7 MongoDB

MongoDB — документо-орієнтована система керування базами даних, яка не потребує опису схеми таблиць. Документо-орієнтована система керування базами даних - це система керування базами даних, спеціально призначена для зберігання ієрархічних структур даних (документів) і зазвичай реалізована за допомогою підходу NoSQL. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними базами даних, функціональними і зручними у формуванні запитів.

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих об'єктів. Різниця між SQL і NoSQL - реляційна база даних (SQL) – база, у котрій дані зберігаються у форматі таблиць, вони суворо структуровані та пов’язані одна з одною. У таблиці є рядки (rows) та стовпчики (columns), кожен є записом, а стовпчик – поле з призначеним йому типом даних. У кожній комірці інформація записана згідно шаблону [8].

Нереляційна база даних (NoSQL) – зберігає дані без чітких зв’язків між собою та без чіткої структури. Замість структурованих таблиць, всередині бази знаходиться безліч різнорідних документів, в тому числі і зображення, відео та навіть публікації у соціальних мережах. На відміну від реляційних БД, NoSQL бази не підтримують SQL запити.

Основні можливості MongoDB:

- Документо-орієнтоване сховище (проста та потужна JSON-подібна схема даних)
- Досить гнучка мова для формування запитів
- Динамічні запити
- Повна підтримка індексів
- Ефективне зберігання бінарних даних великих обсягів, наприклад, фото та відео

MongoDB стала популярною у розробників через свою простоту використання, гнучкість схеми та потужні можливості масштабування. Вона часто використовується для сучасних веб-додатків, де потрібна висока швидкість і можливість легко працювати з напів неструктурованими даними.

2.8 Google API Speech-to-Text

Google API Speech-to-Text - це послуга, що надається Google Cloud, яка дозволяє конвертувати звукові записи (аудіофайли або потоки звукових даних) у текстовий формат за допомогою розпізнавання мовлення. За допомогою цієї API

ви можете легко інтегрувати можливості розпізнавання мовлення безпосередньо у ваші додатки, програми чи веб-сайти. Основні можливості Google API Speech-to-Text розпізнавання мовлення у реальному часі: API може працювати з аудіострімами, що надходять в реальному часі, тобто ви можете отримувати текстові результати практично миттєво під час мовлення. Обробка аудіофайлів: Ви можете завантажувати відео файли у різних форматах на сервери Google і отримувати текстовий вихід. Розпізнавання різних мов: API підтримує багато мов і діалектів, включаючи англійську, іспанську, французьку, німецьку, японську та інші. Адаптація до шуму і акцентів: Система розпізнавання може адаптуватися до шумних умов або особливих акцентів мовця для поліпшення точності розпізнавання [11].

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Процес розробки

Процес розробки варто розпочати з організації створення проекту, організації його базової структури, підключення додаткових бібліотек, модулів та плагінів, а також підключення додаткових технологій, які знадобляться у процесі розробки.

Вікно створення нового проекту у середовищі IntelliJ IDEA можна побачити на рисунку 3.1

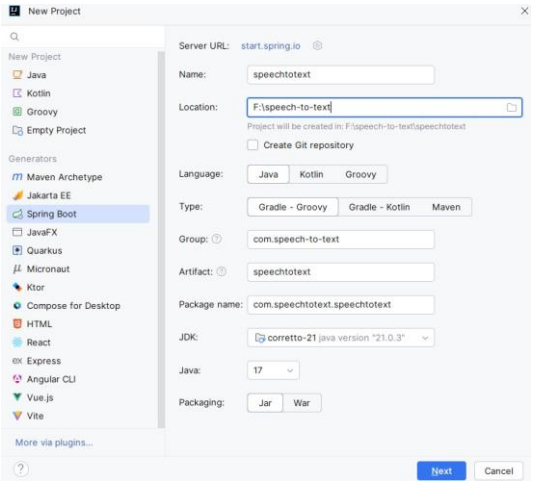


Рисунок 3.1 - Вікно створення нового проекту

Після надання назви, вибору папки, версії Java нам необхідно додати необхідні бібліотеки.

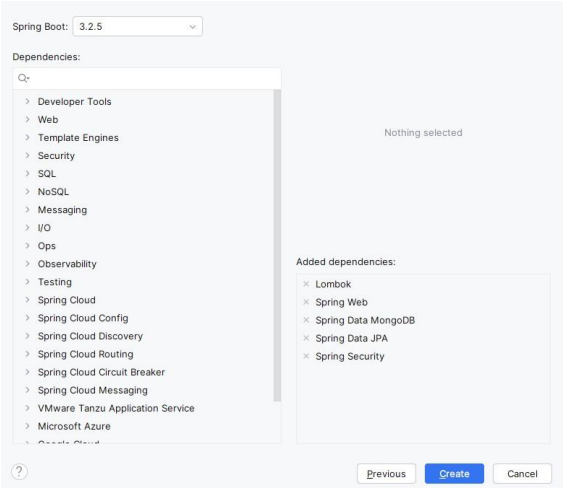


Рисунок 3.2 - Вікно проекту з необхідними бібліотеками

Після вибору версії технології, середовище розробки створить структуру документа, яка зображена на рисунку 3.3

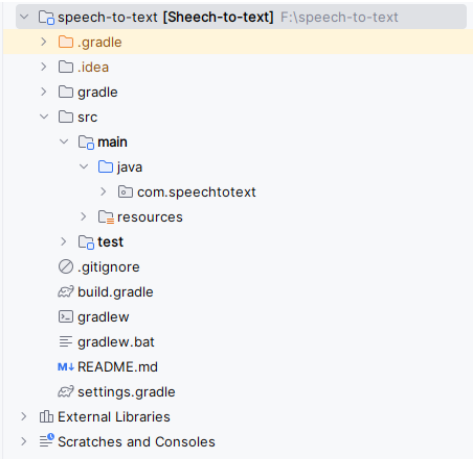


Рисунок 3.3 - Структура нового проекту

Перед початком основної розробки встановимо залежність, яка допоможе у написанні коду для нашого проекту.

ModelMapper - це бібліотека для Java, яка дозволяє легко виконувати взаємне відображення об'єктів одного класу на об'єкти іншого класу. Головна мета ModelMapper - зменшити зусилля, необхідні для ручного написання коду відображення об'єктів вручну.

Щоб додати ModelMapper до вашого проекту, слід виконати наступні кроки:

1. Перейдемо на сайт Maven Repository (https://mvnrepository.com/).
2. У поле пошуку введемо "model mapper".
3. Оберемо перший результат зі списку, який з'явиться в результатах пошуку.
4. Оберемо версію 2.4.3.
5. Скопіюєм нашу залежність і вставимо в наш проеук у build.gradle файл.

```
dependencies { Edit Starters...
    implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'
    implementation 'com.google.cloud:google-cloud-speech:4.25.0'
    implementation 'org.springframework.boot:spring-boot-starter-security'

    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.modelmapper:modelmapper:2.4.3'

    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.junit.jupiter:junit-jupiter-engine:5.9.3'
}
```

Рисунок 3.4 - Встановлення ModelMapper

Тепер давайте підключим базу даних до нашого проекту. Для цього в нашій IDEA виберем закладку Database у правому верхньому куті. Після цього нажмем на +, Data Source і MongoDB. У вікні вводимо назву, пароль, ім'я користувача і клікаємо на Apply.

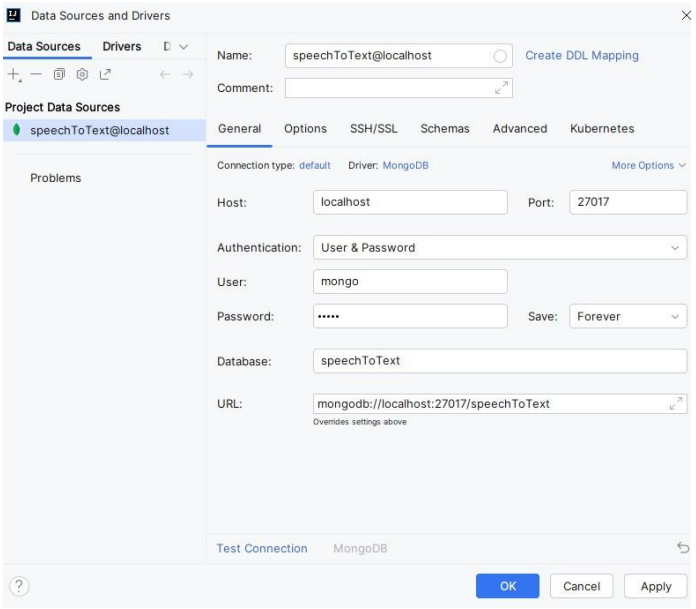


Рисунок 3.5 - Підключення бази даних

Після підключення бази даних, перейдемо до написання коду. Почнемо з нашої моделі.

Для цього в пакеті com.speechtotext створимо пакет models і клас User. В класі User додаєм поля, які ми хочемо додати для нашого користувача це id, email, password, nickName, role. Також слід додати анотації @Document(collection = "users") над класом вказує для якої колекції належить клас User. Анотації @Getter, @Setter зроблять нам методе get, set для зміни і вибору класу User. @NoArgsConstructor @AllArgsConstructor ці анотації створять конструктор з всіма полями і без полів [9]. @Builder - це патерн для ініціалізації об’єкта [1, с.614]. Анотації @Id вказуємо над полем унікального ідентифікатора. @Indexed(unique = true) вказуємо над полем яке буде зберігати тільки унікальні значення. Ідентичну операцію робимо для класу Notes(записки). Створюємо поля id, name, text, date. І не забуваємо про анотації @Getter, @Setter, @NoArgsConstructor, @AllArgsConstructor, @Builder, @Document(collection = "notes"), @Id.

Нижче наведені скріни для класу User і Notes.

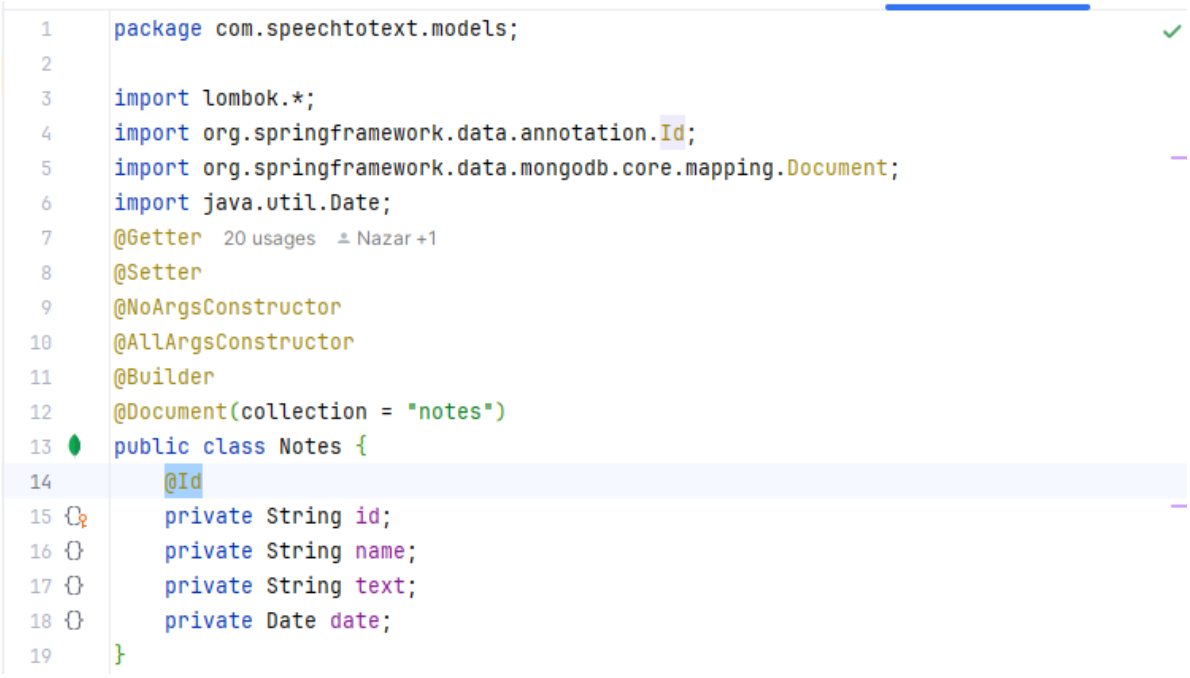
```java
package com.speechtotext.models;

import com.speechtotext.models.enums.Role;
import lombok.*;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.Collection;
import java.util.List;

@Getter  27 usages  ± 2or5 +1 *
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Document(collection = "users")
public class User implements UserDetails {
    @Id
    private String id;
    @Indexed(unique = true)
    private String email;
    private String password;
    private String nickName;
    private Role role;
```

Рисунок 3.6 - Клас User



```java
package com.speechtotext.models;

import lombok.*;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import java.util.Date;
@Getter  20 usages  Nazar +1
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Document(collection = "notes")
public class Notes {
    @Id
    private String id;
    private String name;
    private String text;
    private Date date;
}
```

Рисунок 3.7 - Клас Notes

Після створення класів models, можна створити репозиторії для цих класів. Для цього в нашій папці com.speechtotext створимо пакет під назвою repositories і інтерфейс під назвою UserRepo. В інтерфейсі UserRepo наслідуємося від інтерфейсу MongoRepository. MongoRepository параметризуємо User і String. Це потрібно для того, щоб інтерфейс MongoRepository міг створити базові методи. Над наши інтерфейсом вказуємо анотацію @Repository. Анотацію @Repository - це маркер, щоб Spring знав, що ми хочемо створити об’єкт даного класу [15]. Ще створимо сигнатуру методу під назвою findUserByEmail для того, щоб ми в майбутньому могли шукати користувача за допомоги email. Нагадую, що Spring Data Jpa допомагає нам створити прості методи без їхньої реалізації. Для репозиторію Notes ми робимо аналогічні дії.

```java
package com.speechtotext.repositories;

import com.speechtotext.models.User;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository 8 usages  Nazar +1
public interface UserRepo extends MongoRepository<User,String> {
    Optional<User>findUserByEmail(String username); 4 usages  2or5
}
```

Рисунок 3.8 - UserRepo

```java
package com.speechtotext.repositories;

import com.speechtotext.models.Notes;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface NoteRepo extends MongoRepository<Notes, String> { 2 usages  Nazar
}
```

Рисунок 3.9 NotesRepo

Тепер можна почати працювати з service шаром. В пакеті com.speechtotext створимо пакет service і клас UserServiceImpl. Service - це шар для роботи бізнес логіки, тобто стараємося писати суто java код [15]. Зразу додамо анотацію иаркер Service і поле userRepo з анотацією @AllArgsConstructor для того, що ми могли звертатися до нашого репозиторію. В класі UserServiceImpl зробимо три методи для пошуку всіх користувачі, користувача по id, видалення користувача.

Створюємо метод getAllUsers в якому ми звертаємося до userRepo і дістаємо всіх користувачів з нашої бд. Повертаємо сторінку користувачів для майбутньої пагінації.

Метод getUserById який приймає id, і за допомоги id повернемо користувача, якщо користувача не існує за вказаним id відбудеться помилка з повідомленням "User not found by this id". Для того щоб створити помилку ми в

пакеті com.speechtotext створимо пакет exceptions і клас WrongIdException, який наслідується від RuntimeException. І додаємо пустий конструктор для повідомлення.

Метод deleteUserById за допомоги цього методу буде відбуватися видалення користувача по id. Завдяки userRepo ми видалимо користувача з нашої бд. І повернемо id користувача, якого видалили.

```
import java.util.List;
@Service  ± Nazar +1 *
@AllArgsConstructor
public class UserServiceImpl implements UserService {

    private final UserRepo userRepo;

    @Override  1 usage  ± 2or5
    public List<User> getAllUsers(Pageable pageable) {
        Page<User> page = userRepo.findAll(pageable);
        return page.getContent();
    }

    @Override  1 usage  ± Nazar
    public User getUserById(String Id) {
        return userRepo.findById(Id)
                .orElseThrow(() -> new WrongIdException(ErrorMessages.USER_NOT_FOUND_BY_ID + Id));
    }

    @Override  1 usage  ± Nazar
    public String deleteUserById(String Id) {
        userRepo.deleteById(Id);
        return Id;
    }
```

Рисунок 3.10 - Клас UserServiceImpl

Для NoteServiceImp принцип роботи схожий тільки додамо додаткові методи для роботи з аудіо.

Метод збереження запису в бд дамо назву saveNotes в якому, зробим перевірку по email, та збережемо запис. Але для того щоб зберегти наш запис у формі тексту нам знадобиться додаткова бібліотека Google Cloud Speech to Text. Додаємо в наш файл build.gradle залежність “implementation group: 'com.google.cloud', name: 'spring-cloud-gcp-starter-storage', version: '4.8.4'”. Після цього ми зможем користуватися додатковими класами для перетворення аудіо в

текст. Створимо метод в якому вкажемо формат запису та мову та додаткові перевірки. Нижче наведені скріни методів які допоможуть зберегти наш перетворений запис у базу даних.

```
@Override 1 usage  ± 2or5 +1
public void saveNotes(NotesDto notesDto) {
    User user = userRepo.findUserByEmail(notesDto.getEmail())
            .orElseThrow(()-> new WrongIdException(ErrorMessages.USER_NOT_FOUND_BY_EMAIL));
    Notes notes = Notes.builder()
            .name(notesDto.getName())
            .date(Timestamp.valueOf(LocalDateTime.now()))
            .text(convertAudioToText(notesDto.getBase64()))
            .build();
    noteRepo.save(notes);
    if (user.getNotes() == null) {
        user.setNotes(new ArrayList<>());
    }
    user.getNotes().add(notes);
    userRepo.save(user);
}
```

Рисунок 3.11 - Метод для збереження перетвореного запису в базу даних

```
private String convertAudioToText(String base64) { 1 usage  ± 2or5
    saveAudioOnGoogleCloudBucket(base64);
    try (SpeechClient speechClient = SpeechClient.create()) {
        String gcsUri = "gs://wgebrehnbrethnj4retn/record.mp3";
        RecognitionConfig config =
                RecognitionConfig.newBuilder()
                        .setEncoding(RecognitionConfig.AudioEncoding.MP3)
                        .setSampleRateHertz(16000)
                        .setLanguageCode("en-US")
                        .build();
        RecognitionAudio audio = RecognitionAudio.newBuilder().setUri(gcsUri).build();
        RecognizeResponse response = speechClient.recognize(config, audio);
        List<SpeechRecognitionResult> results = response.getResultsList();

        for (SpeechRecognitionResult result : results) {
            SpeechRecognitionAlternative alternative = result.getAlternativesList().get(0);
            return alternative.getTranscript();
        }
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    return ErrorMessages.CANT_TRANSLATE_AUDIO;
}
```

Рисунок 3.12 - Метод для перетворення запису в текст

Окрім методів збереження, пошуку записів по id, виведення всіх записів, видалення запису, додамо методи editNotes(виправлення записів) і getAllNotesByUserEmail(виведення всіх записів по пошті користувача). В методі editNotes за допомоги userRepo знайдемо записку по id і за допомоги

modelMapper, який ми додали раніше будем зберігати нові дані для записки. Метод getAllNotesByUserEmail, ми зновуштаки за допомоги userRepo знаходим користувача по email і повертає всі записи користувача за певни email.

```
public void editNotes(NotesDto notesDto) {
    Notes note = noteRepo.findById(notesDto.getId())
            .orElseThrow(()-> new WrongIdException(ErrorMessages.NOTE_NOT_FOUND_BY_ID));
    modelMapper.map(notesDto, note);
    noteRepo.save(note);
}

public List<Notes> getAllNotesByUserEmail(String email) {  1 usage   2or5 +1
    User user = userRepo.findUserByEmail(email)
            .orElseThrow(()-> new WrongIdException(ErrorMessages.USER_NOT_FOUND_BY_EMAIL));
    return user.getNotes();
}
```

Рисунок 3.13 - Метод editNotes та getAllNotesByUserEmail

```
public class WrongIdException extends RuntimeException{  11 usages   Nazar
    public WrongIdException(String messages){  7 usages   Nazar
        super(messages);
    }
}
```

```
public class ErrorMessages {  12 usages   Nazar +1
    public static final String USER_NOT_FOUND_BY_ID = "User not found by this id";  1 usage
    public static final String NOTE_NOT_FOUND_BY_ID = "Note not found by this id";  2 usages
    public static final String CANT_TRANSLATE_AUDIO = "Can`t translate your audio";  1 usag
    public static final String USER_NOT_FOUND_BY_EMAIL = "User not found by this email";
    private ErrorMessages() {  no usages   Nazar
    }
```

Рисунок 3.14 - Клас WrongIdException та повідомлення до нього

Також, не забуваємо додати анотації @Service для позначення сервіс класу та анотацію @AllArgsConstructor для полів NoteRepo, ModelMapper, UserRepo, щоб ми змогли викликати методи цих класів.

```
@Service   2or5 +1
@AllArgsConstructor
public class NoteServiceImp implements NoteService {

    private final NoteRepo noteRepo;
    private final ModelMapper modelMapper;
    private final UserRepo userRepo;
```

Рисунок 3.15 - Залежності для класу NoteServiceImp

Залишилось написати фінальний шар Controller. Controller - відповідає за обробку вхідних запитів від користувачів і виконання необхідних дій для відповіді на ці запити. Отже, в пакеті com.speechtotext створимо пакет controllers та клас UsersControllers. Над класом вказуємо анотацію @RestController, @RequestMapping, @AllArgsConstructor.

RestController - це анотація в Spring Framework для позначення класів, які обробляють запити HTTP. У Spring ця анотація дозволяє вам швидко створювати веб-контролери, які обробляють HTTP-запити і повертають дані у форматі, зрозумілому для клієнтів (наприклад, JSON) [15].

@RequestMapping - це анотація в Spring Framework, яка використовується для вказівки маппінгу URL-адреси на метод контролера. Ця анотація дозволяє визначити, як HTTP-запити, отримані певним URL-адресою, будуть оброблятися в вашому додатку [6].

@AllArgsConstructor - ця анотація потрібна для того, щоб внедрити методи клас UserService, які ми створили раніше [9].

Слід зазначити, що над кожним контролером потрібно позначити HTTP методи. Над контролером getAllNotes(вивід список всіх користувачів) вказуємо анотацію @GetMapping("/users"), контролер getUserById (вивід користувача по id) вказуємо анотацію @GetMapping("/{Id}") та контролер deleteUser (видалення користувача по id) вказуємо анотацію @DeleteMapping("/delete-user/{Id}").

Вказуємо анотацію @PathVariable в аргументи метода для того, щоб ми змогли дістати id користувача по url адресі. Напишем ResponseEntity.status (HttpStatus.OK) при випадку, якщо вся операція даного методу пройша успішна, та викличем метод userService.

Аналогічні дії ми робимо для NotesController. Вказуємо над класом анотації @RestController, @RequestMapping("/notes"), @AllArgsConstructor. Прописуємо анотації @GetMapping над методом getAllNotes(вивід список всіх записок), getNoteById(вивід записки по id), getAllNotesByUserEmail(всі записки користувача) та анотація @DeleteMapping над методом deleteNote(видалення

записки по id). Крім цього потрібно вказати анотації @PutMapping, @PostMapping для методів editNote(оновлення даних для записки), createNote(створення записки). Post - використовується для відправки даних на сервер для створення ресурсу. Put - використовується для оновлення існуючого ресурсу або створення нового, якщо ресурс не існує.

```java
@RestController  Nazar +1
@RequestMapping("/user")
@AllArgsConstructor
public class UsersControllers {

    private final UserService userService;

    @GetMapping("/users")  2or5
    public ResponseEntity<List<User>> getAllNotes(Pageable pageable) {
        List<User> users = userService.getAllUsers(pageable);
        return ResponseEntity.status(HttpStatus.OK).body(users);
    }

    @GetMapping("/{Id}")  Nazar
    public ResponseEntity<User> getUserById(@PathVariable String Id) {
        return ResponseEntity.status(HttpStatus.OK).body(userService.getUserById(Id));
    }

    @DeleteMapping("/delete-user/{Id}")  Nazar
    public ResponseEntity<String> deleteUser(@PathVariable String Id){
        return ResponseEntity.status(HttpStatus.OK).body(userService.deleteUserById(Id));
    }
```

Рисунок 3.16 - Клас UsersControllers

```java
@RestController  Nazar +1
@RequestMapping("/notes")
@AllArgsConstructor
public class NotesController {

    private final NoteService noteService;
    @GetMapping  2or5 +1
    public ResponseEntity<List<Notes>> getAllNotes(Pageable pageable) {
        List<Notes> notes = noteService.getAllNotes(pageable);
        return ResponseEntity.status(HttpStatus.OK).body(notes);
    }

    @GetMapping("/note/{id}")  2or5
    public ResponseEntity<NoteDtoResponse> getNoteById(@PathVariable String id){
        return ResponseEntity.status(HttpStatus.OK).body(noteService.getNoteById(id));
    }

    @PostMapping("/create-note")  Nazar
    public ResponseEntity<Notes> createNote(@RequestBody NotesDto notesDto){
        noteService.saveNotes(notesDto);
        return ResponseEntity.status(HttpStatus.CREATED).build();
    }

    @PutMapping("/edit-note")  2or5
    public ResponseEntity<Notes> editNote(@RequestBody NotesDto notesDto){
        noteService.editNotes(notesDto);
        return ResponseEntity.status(HttpStatus.OK).build();
    }

    @GetMapping("/notes-for-user")  2or5
    public ResponseEntity<List<Notes>> getAllNotesByUserEmail(@RequestParam String email){
        return ResponseEntity.status(HttpStatus.OK).body(noteService.getAllNotesByUserEmail(email));
    }

    @DeleteMapping("/delete-note/{id}")  Nazar
    public ResponseEntity<String> deleteNote(@PathVariable String id){
        return ResponseEntity.status(HttpStatus.OK).body(noteService.deleteNote(id));
    }
}
```

Рисунок 3.17 - Клас NotesController

В деяких методах нашого класу NotesController в аргументах є NotesDto. Що таке Dto? DTO (Data Transfer Object) - це шаблон проектування в програмуванні, який використовується для передачі даних між підсистемами програми [15]. У контексті розробки програмного забезпечення, особливо веб-програмування і розробки API, DTO використовується для ізоляції даних, які передаються між компонентами програми, і відділення цих даних від доменних об'єктів або сутностей бази даних.

```
1   package com.speechtotext.DTO;
2
3   import lombok.Getter;
4
5   @Getter  9 usages  2or5 +1
6   public class NotesDto {
7       private String id;
8       private String name;
9       private String text;
10      private String email;
11      private String base64;
12  }
13
```

Рисунок 3.18 - Клас NotesDto

Після створення репозиторіїв, контролерів, сервісу перейдем до авторизації та аутентифікації користувача. Для цього нам знадобиться Spring Security(який підключили раніше) та додаткові залежності, які потрібно додати. В файл build.gradle додаємо implementation 'io.jsonwebtoken:jjwt-api:0.11.2' runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.2' runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.11.2' на рисунку 3.19

```
implementation 'io.jsonwebtoken:jjwt-api:0.11.2'
runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.2'
runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.11.2'
```

Рисунок 3.19 - Файл build.gradle з додатковими залежностями

Після додавання додаткових бібліотек ми можемо використовувати додаткові класи та перейти до створення security config.

Створимо новий пакет під назвою security в якому будуть три класса. Перший клас JwtAuthenticationFilter відповідає за аутентифікації користувача. В цьому класі метод який відповідає за перевірку ролі, валідного токена для входу.

```
package com.speechtotext.config.security;
...

import java.io.IOException;

@Component  2or5
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;
    @Override  no usages  2or5
    protected void doFilterInternal(
            @NonNull HttpServletRequest request,
            @NonNull HttpServletResponse response,
            @NonNull FilterChain filterChain
    ) throws ServletException, IOException {
        final String authHeader = request.getHeader( s: "Authorization");
        final String jwt;
        final String userEmail;
        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }
        jwt = authHeader.substring( beginIndex: 7);
        userEmail = jwtService.extractUsername(jwt);
        if (userEmail != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = this.userDetailsService.loadUserByUsername(userEmail);
            if (jwtService.isTokenValid(jwt, userDetails)) {
                UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
                        userDetails,
                        credentials: null,
                        userDetails.getAuthorities()
                );
                authToken.setDetails(
                        new WebAuthenticationDetailsSource().buildDetails(request)
                );
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        filterChain.doFilter(request, response);

    }
}
```

Рисунок 3.20 - Клас JwtAuthenticationFilter

Другий клас JwtService відповідає за генерацію JWT. JWT (JSON Web Token) - це стандарт, що визначає компактний, самостійний і захищений спосіб передачі інформації між сторонами у вигляді об'єкта JSON. JWT використовується для автентифікації та авторизації. В цьому класі описано логіку створення токену, час скільки дійсний токен та інші класи для перевірки токену на валідність. Токен кодується за допомоги SECRET_KEY [13].

```
package com.speechtotext.config.security;

import ...

@Service  3 usages  2or5 +1
public class JwtService {

    private static final String SECRET_KEY = "404E635266556A586E3272357538782F413F4428472B4B6250645367566B5970";
    private static final String CLAIM_KEY_ROLE = "role";  1 usage

    public String extractUsername(String token) {  2 usages  2or5
        return extractClaim(token, Claims::getSubject);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {  2 usages  2or5
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    public String generateToken(UserDetails userDetails) {  2 usages  Nazar +1
        Map<String, Object> extraClaims = new HashMap<>();
        if (userDetails instanceof User) {
            User user = (User) userDetails;
            extraClaims.put(CLAIM_KEY_ROLE, user.getRole().name());
        }
        return generateToken(extraClaims, userDetails);
    }


    public String generateToken(Map<String, Object> extraClaims, UserDetails userDetails) {  1 usage  2or5
        return Jwts
                .builder()
                .setClaims(extraClaims)
                .setSubject(userDetails.getUsername())
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 24))
                .signWith(getSignInKey(), SignatureAlgorithm.HS256)
                .compact();
    }
```

```
public boolean isTokenValid(String token, UserDetails userDetails) {  1 usage   2or5
    final String username = extractUsername(token);
    return (username.equals(userDetails.getUsername())) && !isTokenExpired(token);
}

private boolean isTokenExpired(String token) { return extractExpiration(token).before(new Date()); }

private Date extractExpiration(String token) { return extractClaim(token, Claims::getExpiration); }

private Claims extractAllClaims(String token) {  1 usage   2or5
    return Jwts
            .parserBuilder()
            .setSigningKey(getSignInKey())
            .build()
            .parseClaimsJws(token)
            .getBody();
}

private Key getSignInKey() {  2 usages   2or5
    byte[] keyBytes = Decoders.BASE64.decode(SECRET_KEY);
    return Keys.hmacShaKeyFor(keyBytes);
}
```

Рисунок 3.21 - Клас JwtService

Третій клас SecurityConfig об'єднує попередні два класса та вказуємо який функціонал може використовувати користувач з ролю ADMIN та USER. Клас позначимо анотацією @Configuration. Аннотація @Configuration позначає клас як конфігураційний клас Spring, який визначає біни (компоненти), що використовуються в програмі. Вказуємо ендпоїнти які доступні тільки для користувача з ролю ADMIN та ендпоїнти які доступні тільки для користувача з ролю USER. Метод filterChain з анотацією @Bean: Цей метод визначає конфігурацію безпеки для додатку за допомогою Spring Security. Параметр HttpSecurity http представляє об'єкт конфігурації безпеки HTTP. В методі filterChain виконуються налаштування для Налаштування прав доступу для різних URL шляхів за допомогою authorizeHttpRequests. Додавання надавача аутентифікації (authenticationProvider), який буде використовуватися для проведення аутентифікації. Додавання фільтра jwtAuthFilter до ланцюгу фільтрів перед фільтром UsernamePasswordAuthenticationFilter, що дозволяє обробляти JWT токени для аутентифікації. Метод повертає об'єкт

SecurityFilterChain, який представляє ланцюг фільтрів безпеки для застосунку, побудовану на основі налаштувань Security. Отже, цей клас SecurityConfig встановлює конфігурацію безпеки для застосунку, включаючи обробку JWT токенів, налаштування прав доступу до ресурсів і визначення стратегій керування сеансами [13].

```java
package com.speechtotext.config.security;

import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import static com.speechtotext.models.enums.Role.ADMIN;
import static com.speechtotext.models.enums.Role.USER;

@Configuration  2or5 +1
@RequiredArgsConstructor
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthFilter;
    private final AuthenticationProvider authenticationProvider;

    @Bean  2or5 +1
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
                .csrf(AbstractHttpConfigurer::disable)
                .authorizeHttpRequests(req ->
                        req.requestMatchers("/notes/**", "/auth/**").permitAll() AuthorizationManagerRequestMa
                                .requestMatchers("/user/**").hasAuthority("ADMIN")
                                .anyRequest() AuthorizedUrl
                                .authenticated()
                )
                .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authenticationProvider(authenticationProvider)
                .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

}
```

Рисунок 3.22 - Клас SecurityConfig

Проект з сторони серверу готовий. Можна подивитися чи все коректно працює за допомоги програми postman. Для цього спробуємо зареєструвати нового користувача та створити записку. На рисунку 3.23 кидаємо запрос по нашій url реєстрації з тілом запросу, в якому вказано пошту, нікнейм, пароль та дивимось зареєстрованого користувача у базі даних. На рисунку 3.24 створюємо записку в якому вказано base64(запис у закодованому вигляді), ім’я записки та користувача.

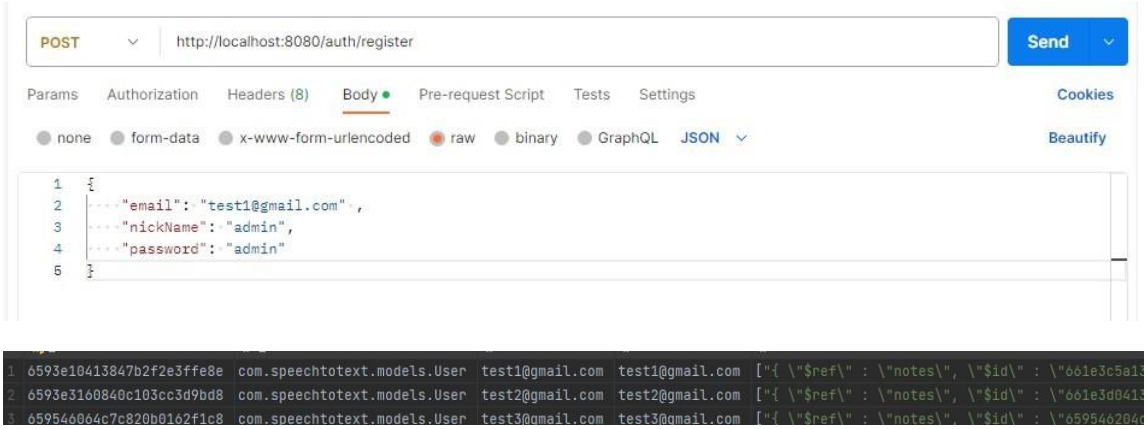
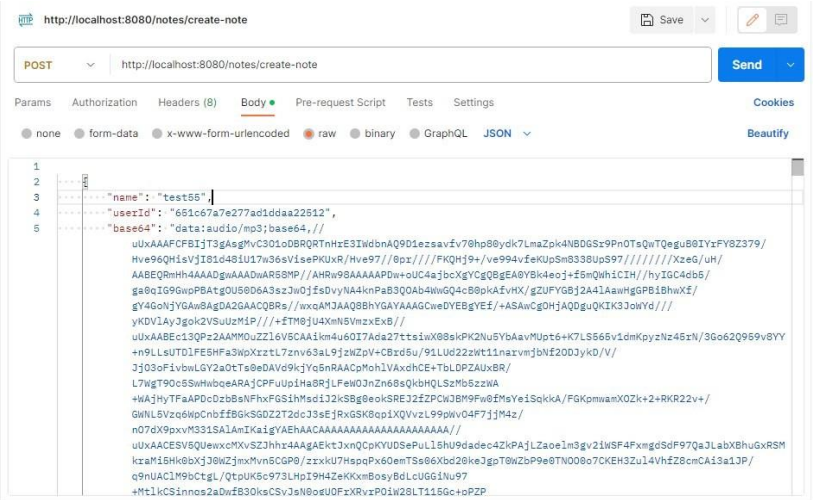


Рисунок.3.23 - Збережений користувач



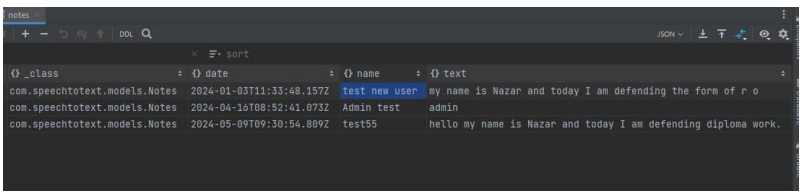


Рисунок 3.24 - Збережена записка

3.2 Інтерфейс клієнтської частина

Головна сторінка сайту має сучасний зручний лаконічний інтерфейс, який привертає увагу користувача, але не перевантажувати його увагу надмірним функціоналом.

Почнемо з додавань потрібних залежностей за допомоги команди npm install. На рисунку 3.25 зображено залежності, які потрібні для подальшої роботи.

```
{
  "name": "speeech-gold-v1",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@fortawesome/fontawesome-svg-core": "^6.5.0",
    "@fortawesome/free-solid-svg-icons": "^6.5.0",
    "@fortawesome/react-fontawesome": "^0.2.0",
    "@mui/material": "^5.14.18",
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.6.2",
    "bootstrap": "^5.3.2",
    "date-fns": "^3.0.5",
    "mic-recorder-to-mp3": "^2.2.2",
    "react": "^18.2.0",
    "react-bootstrap": "^2.9.1",
    "react-dom": "^18.2.0",
    "react-material-ui-carousel": "^3.4.2",
    "react-player": "^2.13.0",
    "react-router-dom": "^6.20.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
```

Рисунок 3.25 - Залежності

Після додавання залежностей, реалізуємо NavigationBar для комфортного переходу між сторінками. На рисунку 3.26 зробажений файла App.js в якому код та посилання при переході на певну сторінку.

```
 9  import NoteEdit from "./components/Note/NoteEdit";
10  import UserList from "./components/User/UserList";
11  import Register from "./components/User/Register";
12  import Login from "./components/User/Login";
13  import NoteListForUser from "./components/Note/NoteListForUser";
14
15  export default function App() {
16    return (
17      <Router>
18        <NavigationBar />
19        <Container>
20          <Row>
21            <Col lg={12}>
22              <Routes>
23                <Route path="/" element={<Welcome />} />
24                <Route path="/add" element={<Note />} />
25                <Route path="/edit/:id" element={<NoteEdit />} />
26                <Route path="/list" element={<NoteList />} />
27                <Route path="/users" element={<UserList />} />
28                <Route path="/register" element={<Register />} />
29                <Route path="/login" element={<Login />} />
30                <Route path="/list-user-notes" element={<NoteListForUser />} />
31              </Routes>
32            </Col>
33          </Row>
34        </Container>
35        <Footer />
36      </Router>
```

Рисунок 3.26 - Файл App.js

Для реалізації перегляду, видалення, редагування та створення записок створимо компоненти під назвою Note.js, NoteList.js. У додатку Б код для компонент Note.js в якому знаходиться функції початок запису, зупинка запису, кодування в base64 та повертає зручну форму користувача. На рисунку 3.27 зображено компонент NoteList.js який повертає таблицю з записками та кнопкою видалення та редагування вибраної записки [14].

```
      }
    });
  };

  changePage = (event) => {
    this.setState({
      [event.target.name]: parseInt(event.target.value),
    });
  };

  firstPage = () => {
    if (this.state.currentPage > 1) {
      this.setState({
        currentPage: 1,
      });
    }
  };

  prevPage = () => {
    if (this.state.currentPage > 1) {
      this.setState({
        currentPage: this.state.currentPage - 1,
      });
    }
```

```
  };

  lastPage = () => {
    if (
      this.state.currentPage <
      Math.ceil(this.state.notes.length / this.state.notesPerPage)
    ) {
      this.setState({
        currentPage: Math.ceil(
          this.state.notes.length / this.state.notesPerPage
        ),
      });
    }
  };

  nextPage = () => {
    if (
      this.state.currentPage <
      Math.ceil(this.state.notes.length / this.state.notesPerPage)
    ) {
      this.setState({
        currentPage: this.state.currentPage + 1,
      });
    }
```

```
render() {
  const { notes, currentPage, notesPerPage } = this.state;
  const lastIndex = currentPage * notesPerPage;
  const firstIndex = lastIndex - notesPerPage;
  const currentNotes = notes.slice(firstIndex, lastIndex);
  const totalPages = notes.length / notesPerPage;

  return (
    <div>
      <div style={{ display: this.state.show ? "block" : "none" }}>
        <CreateNoteToast
          children={{
            show: this.state.show,
            message: "Note Deleted Successfully.",
            type: "danger",
          }}
        />
      </div>
      <Card
        className="border border-dark bg-dark text-white"
        style={{ marginTop: "37px" }}
      >
        <Card.Header>
          <FontAwesomeIcon icon={faList} /> Your Notes
        </Card.Header>
        <Card.Body>
```

Рисунок.3.27 - Компонент NoteList.js

Після створення двох компонентів створимо форму для реєстрації та логінації для користувача. На рисунку 3.28 зображено код для реєстрації користувача. На рисунку 3.29 зображено код для логінації.

```
                />
              </InputGroup>
            </Form.Group>
            <Form.Group style={{ marginTop: "15px" }}>
              <InputGroup>
                <InputGroup.Text>
                  <FontAwesomeIcon icon={faLock} />
                </InputGroup.Text>
                <FormControl
                  required
                  autoComplete="off"
                  type="password"
                  name="password"
                  value={password}
                  onChange={handleChange}
                  placeholder="Enter Password"
                />
              </InputGroup>
            </Form.Group>
          </Card.Body>
          <Card.Footer style={{ textAlign: "right" }}>
          <Link to="/login">
            <Button variant="light">
            <FontAwesomeIcon icon={faSignInAlt} /> Login
            </Button>
          </Link>{' '}
            <Button type="submit" variant="light" disabled={email.length === 0 || password.length === 0 || nickName.lengt
              <FontAwesomeIcon icon={faRegistered} /> Register
            </Button>
          </Card.Footer>
        </Form>
      </Card>
    </Col>
  </Row>
  );
};

export default Register;
```

Рисунок 3.28 - Register.js

```
            </Card.Body>
            <Card.Footer style={{ textAlign: "right" }}>
            <Link to="/register">
              <Button variant="light">
              <FontAwesomeIcon icon={faRegistered} /> Register
              </Button>
            </Link>{' '}
              <Button type="submit" variant="light" disabled={email.length === 0 || password.length === 0}>
                <FontAwesomeIcon icon={faSignInAlt} /> Login
              </Button>
            </Card.Footer>
          </Form>
        </Card>
      </Col>
    </Row>
  );
};

export default Login;
```

Рисунок 3.29 - Login.js

Наша фронтенд частина готова. За допомоги команди npm start запускаєм проект і перевіряємо роботу.

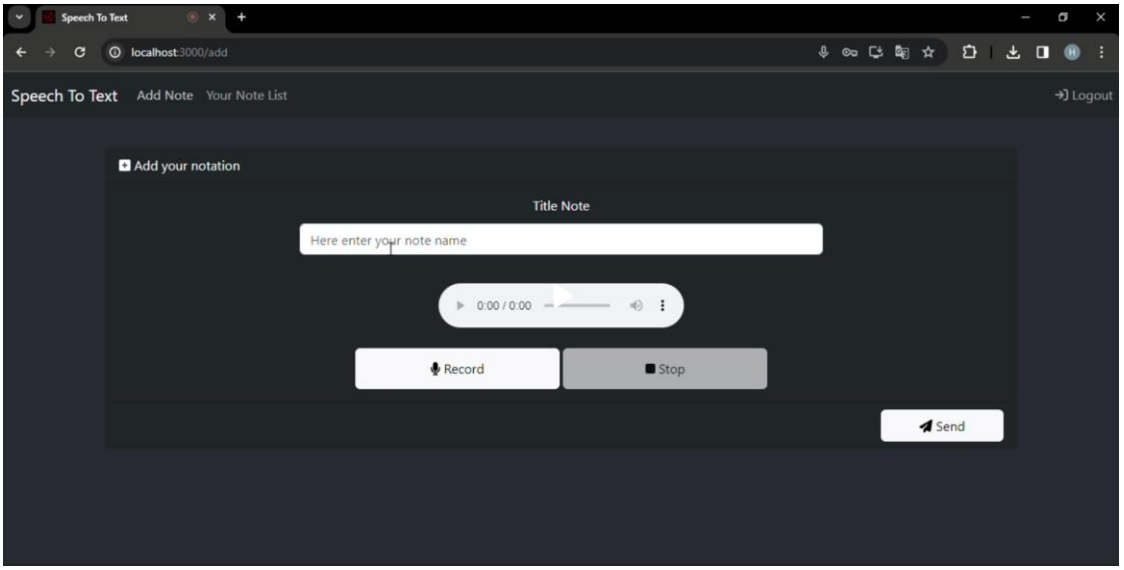


Рисунок 3.30 - Інтерфейс при створенні записки

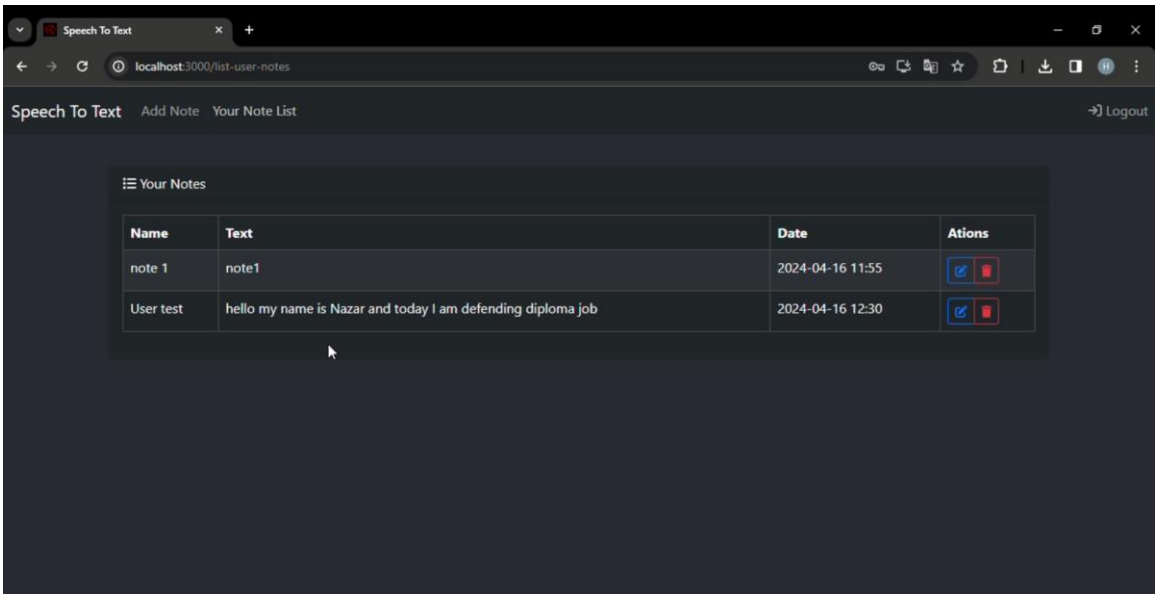


Рисунок 3.31 - Інтерфейс при перегляді записок

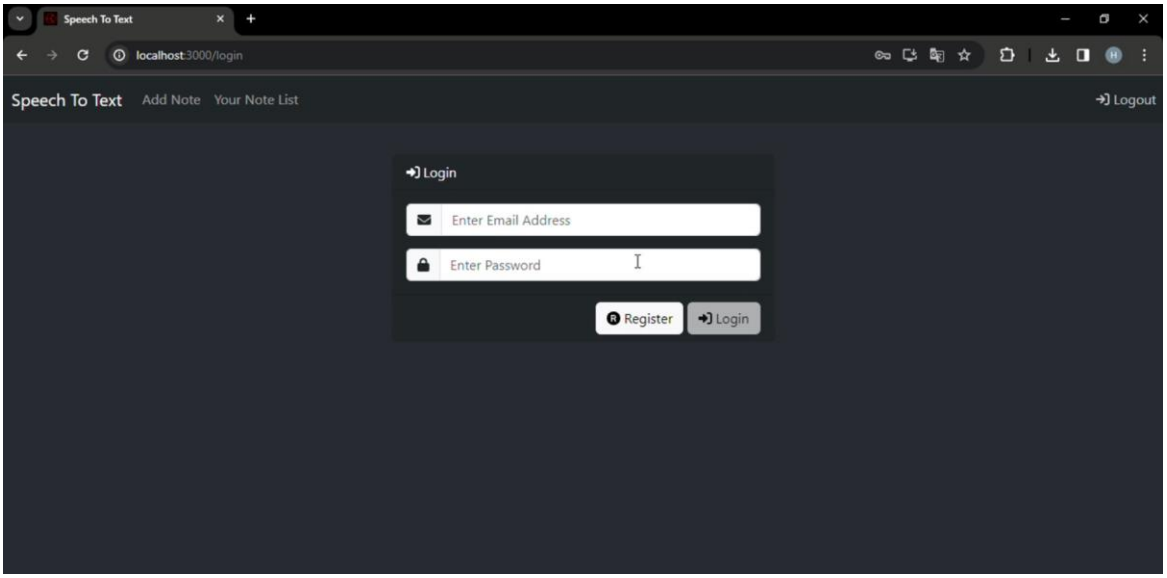


Рисунок 3.32 - Інтерфейс логінації

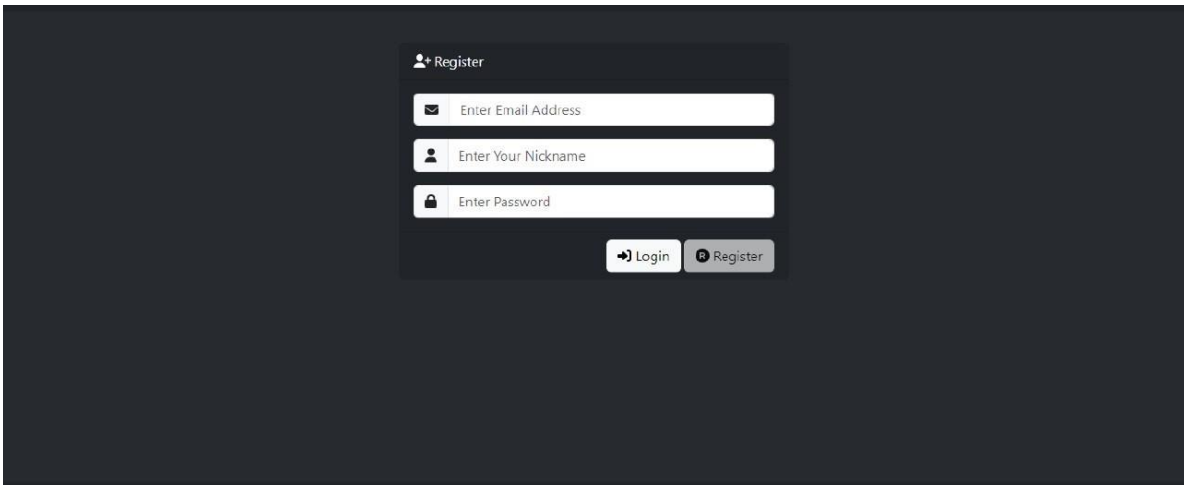


Рисунок 3.33 - Інтерфейс реєстрації

ВИСНОВКИ

У даній дипломній роботі був розроблений веб-додаток, який використовує технології Spring Boot, Google Cloud Speech-to-Text, Java, React, MongoDB, Spring Security та інші технології, для перетворення аудіофайлів у текстовий формат.

Під час розробки додатку було проведено дослідження та аналіз сучасних технологій автоматичного розпізнавання мови, що дало змогу обрати оптимальний інструментарій для досягнення поставленої мети. Використання Spring Boot спростило створення веб-інтерфейсу для користувачів, забезпечуючи зручний доступ до функціоналу додатку через веб-браузер. У процесі розробки було реалізовано наступні ключові функціональні можливості: перетворення аудіо у текст за допомогою Google Cloud Speech-to-Text API, відображення результатів у веб-інтерфейсі користувача.

У подальших дослідженнях та розвитку проекту можна розглядати вдосконалення алгоритмів розпізнавання мови, підтримку більшої кількості мов та вдосконалення інтерфейсу користувача для забезпечення більшої зручності та функціональності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Патерни проектування: Ерік Фрімен, Елізабет Робсон - Листопад 30, 2020.
2. Ефективне програмування 3-тє видання: Джошуа Блох - Грудень 27, 2017.
3. Чистий код: Роберт Мартін - Лютий 15, 2019.
4. Java. Керівництво для починаючих: Герберт Шилдт - Травня 24, 2020.
5. Spring Security [Електронний ресурс] -Режим доступу до ресурсу: https://spring.io/projects/spring-security (дата звернення: 15.03.2024).
6. Spring Boot [Електронний ресурс]. Режим доступу до ресурсу: https://spring.io/projects/spring-boot (дата звернення: 27.02.2024).
7. Spring Data Jpa [Електронний ресурс]. Режим доступу до ресурсу: https://spring.io/projects/spring-data-jpa (дата звернення: 04.03.2024).
8. MongoDB [Електронний ресурс]. Режим доступу до ресурсу: https://www.mongodb.com/docs/manual/ (дата звернення: 03.03.2024).
9. Lombok [Електронний ресурс]. Режим доступу до ресурсу: https://projectlombok.org/features/ (дата звернення: 17.03.2024).
10. React [Електронний ресурс]. Режим доступу до ресурсу: https://react-bootstrap.netlify.app/docs/components (дата звернення: 20.03.2024).
11. Speech to text [Електронний ресурс]. Режим доступу до ресурсу: https://cloud.google.com/speech-to-text (дата звернення: 10.02.2024).
12. Hibernate [Електронний ресурс]. Режим доступу до ресурсу: https://hibernate.org/orm/documentation/6.5/ (дата звернення: 07.03.2024).
13. YouTube [Електронний ресурс]. Режим доступу до ресурсу: https://www.youtube.com/@amigoscode (дата звернення: 15.03.2024).
14. w3school [Електронний ресурс]. Режим доступу до ресурсу: https://www.w3schools.com/js/js_intro.asp (дата звернення: 19.03.2024).
15. Udemy [Електронний ресурс]. Режим доступу до ресурсу: https://www.udemy.com/course/learn-spring-boot (дата звернення: 11.02.2024).

ДОДАТКИ

Додаток А. Лістинги коду розроблених контролерів

```
package com.speechtotext.controllers;
import com.speechtotext.service.UserService; import
com.speechtotext.models.User;
import lombok.AllArgsConstructor;
import org.springframework.data.domain.Pageable; import
org.springframework.http.HttpStatus; import
org.springframework.http.ResponseEntity; import
org.springframework.web.bind.annotation.*; import java.util.List;

@RestController @RequestMapping("/user")
@AllArgsConstructor
public class UsersControllers {

    private final UserService userService;

    @GetMapping("/users")
    public ResponseEntity<List<User>> getAllNotes(Pageable
pageable) {
        List<User> users = userService.getAllUsers(pageable);
        return ResponseEntity.status(HttpStatus.OK).body(users);
    }
```

```
    @GetMapping("/{Id}")
    public ResponseEntity<User> getUserById(@PathVariable String Id) {
        return ResponseEntity.status(HttpStatus.OK).body(userService.ge
tUserById(Id));
    }

    @DeleteMapping("/delete-user/{Id}") public
    ResponseEntity<String>
deleteUser(@PathVariable String Id){
        return ResponseEntity.status(HttpStatus.OK).body(userService.de
leteUserById(Id));
    }
}

package com.speechtotext.controllers;
import com.speechtotext.DTO.NoteDtoResponse; import
com.speechtotext.DTO.NotesDto;
import com.speechtotext.models.Notes;
import com.speechtotext.service.NoteService; import
lombok.AllArgsConstructor;
import org.springframework.data.domain.Pageable; import
org.springframework.http.HttpStatus; import
org.springframework.http.ResponseEntity; import
org.springframework.web.bind.annotation.*; import java.util.List;
```

```
@RestController @RequestMapping("/notes")
@AllArgsConstructor
public class NotesController {

    private final NoteService noteService; @GetMapping
    public ResponseEntity<List<Notes>> getAllNotes(Pageable pageable) {
        List<Notes> notes = noteService.getAllNotes(pageable);
        return ResponseEntity.status(HttpStatus.OK).body(notes);
    }

    @GetMapping("/note/{id}") public
ResponseEntity<NoteDtoResponse>getNoteById(@PathVariable String id){
returnResponseEntity.status(HttpStatus.OK).body(noteServ ice.getNoteById(id));
    }

    @PostMapping("/create-note")
    public ResponseEntity<Notes> createNote(@RequestBody NotesDto notesDto){
        noteService.saveNotes(notesDto);
        return ResponseEntity.status(HttpStatus.CREATED).build();
    }
```

```
    @PutMapping("/edit-note")
    public ResponseEntity<Notes> editNote(@RequestBody NotesDto notesDto){
        noteService.editNotes(notesDto);
        return ResponseEntity.status(HttpStatus.OK).build();
    }

    @GetMapping("/notes-for-user") public
    ResponseEntity<List<Notes>>
getAllNotesByUserEmail(@RequestParam String email){
        return ResponseEntity.status(HttpStatus.OK).body(noteService.ge
tAllNotesByUserEmail(email));
    }

    @DeleteMapping("/delete-note/{id}") public
    ResponseEntity<String>
deleteNote(@PathVariable String id){
        return ResponseEntity.status(HttpStatus.OK).body(noteService.de
leteNote(id));
    }
}

package com.speechtotext.controllers;
import com.speechtotext.DTO.AuthenticationRequestDto; import
com.speechtotext.DTO.AuthenticationResponseDto; import
com.speechtotext.DTO.RegisterRequestDto;
```

```
import com.speechtotext.service.AuthenticationService;
org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

@RestController @RequestMapping("/auth")
@RequiredArgsConstructor
public class AuthenticationController {

    private final AuthenticationService authenticationService;

    @PostMapping("/register")
    public ResponseEntity<AuthenticationResponseDto>
register(@RequestBody RegisterRequestDto request) {
        return ResponseEntity.ok(authenticationService.register(request
));
    }

    @PostMapping("/authenticate")
    public ResponseEntity<AuthenticationResponseDto>
authenticate(@RequestBody AuthenticationRequestDto request) {
        return ResponseEntity.ok(authenticationService.authenticate(req uest));
    }
}
```

Додаток Б. Лістинги коду Note сторінки

```
  state = { isRecording: false,
      blobUrl: "", isBlocked:
      false,
  };

  submitNote = (event) => { event.preventDefault();

      const note = {
          email: userEmail(), name:
          this.state.title,
          base64: this.state.baseAudio,
      };

      axios
          .post("http://localhost:8080/notes/create-note",
note)
          .then((response) => {
              if (response.data != null) { this.setState({ show: true });
                  setTimeout(() => this.setState({ show: false
}), 6000);
              } else {
                  this.setState({ show: false });
              }
            });
```

```
    this.setState({ title: "",
        isRecording: false,
        blobURL: "",
        baseAudio: "",
    });
};

start = () => {
    if (this.state.isBlocked) { console.log("Permission Denied");
    } else { Mp3Recorder.start()
            .then(() => {
                        this.setState({ isRecording: true });
            })
            .catch((e) => console.error(e));
    }
};

stop = () => { Mp3Recorder.stop()
        .getMp3()
        .then(async ([buffer, blob]) => {
            const blobURL = URL.createObjectURL(blob); const file = new
            File(buffer, "audio.mp3", {
                type: blob.type, lastModified: Date.now(),
```

```
            });
            let baseAudio = await this.audioToBase64(file); console.log("Base audio
            ====>", baseAudio); this.setState({ blobURL, isRecording: false,
baseAudio });
        })
        .catch((e) => console.log(e));
  };

  audioToBase64 = async (audioFile) => { return new
      Promise((resolve, reject) => {
          let reader = new FileReader(); reader.onerror = reject;
          reader.onload = (e) => resolve(e.target.result);
          reader.readAsDataURL(audioFile);
      });
  };
```